UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Servicio cloud para formatos de libros electrónicos en dispositivos de baja resolución

Autor/es

RUBÉN SÁENZ FRANCIA

Director/es

ANA ROMERO IBÁÑEZ

Facultad

Escuela de Máster y Doctorado de la Universidad de La Rioja

Titulación

Máster Universitario en Tecnologías Informáticas

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2016-17

Servicio cloud para formatos de libros electrónicos en dispositivos de baja resolución, de RUBÉN SÁENZ FRANCIA
(publicada por la Universidad de La Rioja) se difunde bajo una Licencia Creative Commons Reconocimiento-NoComercial-SinObraDerivada 3.0 Unported.
Permisos que vayan más allá de lo cubierto por esta licencia pueden solicitarse a los titulares del copyright.

© El autor, 2017
© Universidad de La Rioja, 2017
publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

Trabajo de Fin de Máster

Servicio cloud para formatos de libros electrónicos en dispositivos de baja resolución

Autor:

Rubén Sáenz Francia

Tutor/es: Ana Romero Ibáñez

MÁSTER:
Máster en Tecnologías Informáticas (853M)

Escuela de Máster y Doctorado

AÑO ACADÉMICO: 2016/2017

ÍNDICE

1 RESUMEN EN CASTELLANO Y EN INGLÉS: SÍNTESIS DEL TFM.

Resumen

El objetivo de este proyecto es diseñar y desarrollar una aplicación web que genera libros digitales que contienen problemas matemáticos y desplegar dicha aplicación en Internet en algún Servicio Cloud.

La meta del proyecto es también desacoplar al máximo el código fuente usando alguno de los patrones de diseño más actuales como: Capas de Persistencia usando ORM (Mapeo Objeto-Relacional), patrón Servicio-Repositorio y el Framework Spring MVC (Modelo-Vista-Controlador).

Abstract

The purpose of this project is to design and develop a Web Application that creates digital books which contain mathematical problems and deploy it into Cloud Services on the Internet.

The aim of the project is also to disengage the code using some design patterns such as: Persistence layers using an ORM (Object Relational Mapping), Service - Repository Pattern and the Spring MVC Framework (Model-View-Controller).

2 INTRODUCCIÓN

La idea del proyecto surge a partir de un trabajo que llevo realizando desde el año 2012 y que a grandes rasgos consiste en introducir unos problemas matemáticos en una plataforma informática para la gestión del concurso de primavera de matemáticas que en La Rioja está organizado por la asociación de profesores A-prima.

2.1 Antecedentes

El concurso de Primavera de Matemáticas es una prueba que se realiza anualmente. En ella participan alumnos desde quinto de primaria hasta segundo de bachiller. Los alumnos están separados por cuatro niveles y consta de dos fases:

1. Una fase clasificatoria que se realiza en los Institutos y Colegios. Esta fase se realiza aproximadamente en Febrero. En esta primera fase en 2017 han participado 72 centros con 5675 alumnos.
2. Una segunda fase a la que van los mejores alumnos de cada Instituto y que en La Rioja se celebra aproximadamente en Abril. A esta segunda fase en 2017 se clasificaron 469 alumnos, de los cuales se presentaron 435.

Las pruebas están almacenadas separadas por preguntas en un servidor usando páginas web de hipertexto y las imágenes que dicho archivo usa son almacenadas en otra carpeta llamada graficosp. Este proceso se realiza mediante una conversión semi-automática de un archivo Word que genera todos los archivos necesarios para poder subirlos al servidor.

Para realizar esta conversión semiautomática lo primero que hay que hacer es transformar el archivo .doc que la organización nos envía a otro archivo .doc que tiene unos estilos de párrafo y de imagen específicos. Dicho archivo .doc es guardado como página web filtrada y esa página web filtrada es enviada a un programa informático que realicé en el año 2012 que se encarga por cada prueba de generar tres tipos de contenido:

- Las distintas preguntas de las pruebas almacenadas con extensión htm.

- Las imágenes que contienen dichas preguntas.
- Las inserciones en la base de datos, en formato sql, necesarias para que las pruebas se visualicen.

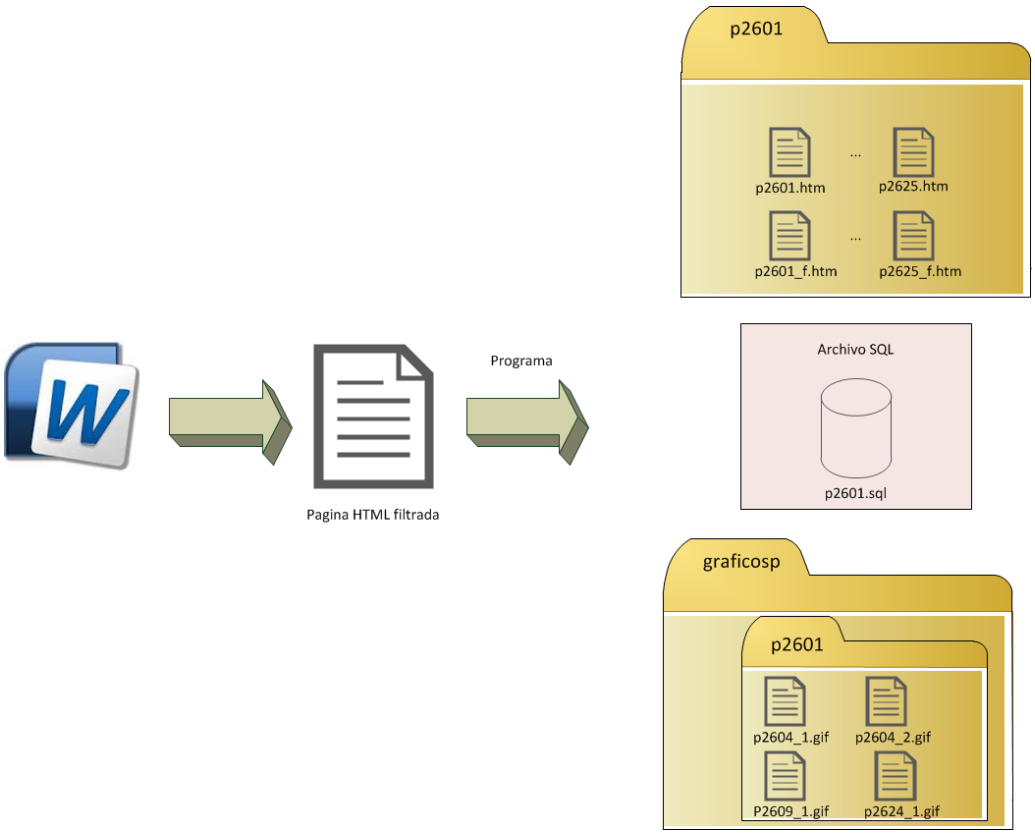


Ilustración 1 - Proceso de generación de los archivos de una prueba

El trabajo que realizo consiste en dos partes:

1. La inserción de 200 preguntas anuales en una aplicación Web desglosadas en cuatro niveles (50 por nivel).
2. La corrección de dichas pruebas usando un lector óptico que la Universidad de La Rioja nos presta.

Para realizar este proyecto vamos a ignorar esta segunda parte que es realizada con otro programa informático que yo realicé y nos vamos a centrar en esta primera parte de tratamiento de las pruebas y de las preguntas que contienen dichas pruebas.

Cada archivo htm generado tiene la siguiente estructura:

```
<div class=PrimaveraEnunciado>TextoPregunta</div>
<div class=PrimaveraRespuestas>
<table class="respuestas"><tr>
    <th id="CA">- A - </th>
    <th id="CB">- B - </th>
    <th id="CC">- C - </th>
    <th id="CD">- D - </th>
    <th id="CE">- E - </th></tr>
```

```
<tr>
    <td id="RA">Respuesta1</td>
    <td id="RB">Respuesta2</td>
    <td id="RC">Respuesta3</td>
    <td id="RD">Respuesta4</td>
    <td id="RE">Respuesta5</td>
</tr></table></div>
```

Como se puede observar las preguntas son siempre de respuesta múltiple con cinco respuestas posibles. Todas las preguntas siguen el mismo formato a pesar de que el peso que se ha dado a las preguntas acertadas, falladas y no contestadas ha ido cambiando con los años.

Los archivos de los que se consta son 3800 archivos de pregunta y 6105 imágenes lo que hacen un total de aproximadamente unos 10000 archivos.

2.2 Alcance

Vista la gran cantidad de preguntas que hay insertadas en el sistema se observa la necesidad de crear un libro electrónico que me permita la lectura de dichas preguntas en cualquier lugar usando como medio de lectura un Ebook.

El formato de creación del libro electrónico va a ser EPUB por ser el más ampliamente utilizado en libros electrónicos. El lector electrónico denominado Kindle que es el que yo tengo no es capaz de leer estos archivos pero existen programas como Calibre que permiten realizar una rápida conversión de los archivos.

Por cada nivel se va a permitir incluir en el libro electrónico el número de fases que queramos y el sistema tiene que servir para en años futuros incorporar las nuevas pruebas que se vayan haciendo de un modo fácil.

2.3 Tecnologías

Durante el proyecto se van a utilizar las siguientes tecnologías:

- **Epublib**: librería java que me permite la creación y lectura de libros Epub.
- **Selenium**: herramienta que me permite ejecutar tests de aceptación de la aplicación.
- **JSP:** me permite crear páginas dinámicas.
- **Thymeleaf:** es un servidor de plantillas que me permite también crear páginas dinámicas. La ventaja que aporta respecto a jsp es que tiene un pseudo-lenguaje xml que me permite la creación de estructuras de control

para mostrar los elementos, lo que me permite desacoplar todavía más el código.

- **Epub**: formato de salida que tendrá el libro electrónico generado.
- **JaCoCo**: la palabra proviene de Java Code Coverage y me permite detectar la cantidad de código que está siendo testado.
- **Jtidy y Jsoup**: me permitirá arreglar el html mal formado.
- **Maven** para la gestión del código fuente del proyecto.
- **JPA**: me permitirá leer/escribir datos en la base de datos de modo sencillo usando unas clases de persistencia que serán generadas mediante las JPA-Tools.
- **JPA-Tools**: herramienta integrada en eclipse que me permitirá obtener las clases de persistencia JAVA leyendo la estructura de una base de datos existente.
- **Plataforma Cloud:** es un paradigma que permite ofrecer servicios de computación a través de Internet. Las ventajas que aporta son:
    - Soporta diferentes entornos y lenguajes de programación.
    - Despliegue rápido.
    - Escalado horizontal y vertical.
    - Ahorro en costes.
    - Gran rendimiento y optimización.
- **Java Beans**: es un estándar que deben cumplir las clases. Entre sus características aparecen:
    - Debe tener un constructor sin argumentos.
    - Sus atributos de clase deben ser privados.
    - Sus propiedades deben ser accesibles mediante métodos get y set que siguen una convención de nomenclatura estándar.
    - Debe ser serializable.
- **Spring-mvc**: framework para desarrollar aplicaciones web basadas en Modelo-Vista-Controlador.

3 ANÁLISIS

Hay un objetivo principal del proyecto que es:

Crear libros electrónicos con formato epub bajo demanda a través de un servicio web, asegurando que los contenidos generados están bien formados.

3.1 Requisitos

Partiendo de este punto podemos generar los diferentes requisitos:

Requisitos Funcionales

R. 1. Se debe generar una aplicación Web que bajo demanda genere libros electrónicos con problemas matemáticos. Que los libros electrónicos se vayan a generar bajo demanda quiere decir que en el momento en el que el usuario lo solicite se va a crear un libro específicamente para ese usuario con la información que ese usuario haya querido incluir.

R. 2. En el libro electrónico generado se debe poder navegar entre las distintas secciones a través de un índice.

R. 3. Desde los problemas se debe poder navegar hacia las soluciones y viceversa.

R. 4. Debido a la gran cantidad de archivos que contiene una Prueba, lo máximo que se va a permitir es la creación de un libro electrónico con todos los enunciados de un determinado nivel.

R. 5. Los niveles que vamos a poder seleccionar para generar el ebook van del 1 al 4.

Requisitos No Funcionales

R. 6. El formato de exportación del R. 1 debe ser Epub.

R. 7. El índice del R. 2 debe contener una marcada diferencia entre los enunciados y sus soluciones.

R. 8. Los problemas y las soluciones del R. 3 deben estar en secciones separadas porque sino nos tentaría a mirar la solución.

R. 9. Los enunciados del libro electrónico generado deben estar adaptados para poder visualizarse correctamente en una pantalla de resolución mínima de 600 píxeles de ancho.

R. 10. La aplicación debe alojarse en una infraestructura cloud.

Objetivos Personales que también debemos satisfacer:

- Considero que un 85% del código generado debe ser comprobado para asegurarnos que todo funciona correctamente. Se ha elegido este número porque llegar a un 100% es prácticamente imposible debido a la existencia en numerosas ocasiones de código defensivo (Ej, los catch) que es difícil de probar.

3.2 Análisis de los archivos que contiene el libro electrónico

EPUB o ePub (acrónimo de la expresión inglesa Electronic publication - Publicación electrónica) es un formato redimensionable de código abierto para leer textos e imágenes.

La ventaja de ePub es que permite redimensionar el tamaño del texto y aplicar distintas tipografías sin necesidad de cambiar los contenidos.

Un fichero EPUB consiste en un archivo comprimido .zip que contiene la siguiente estructura:

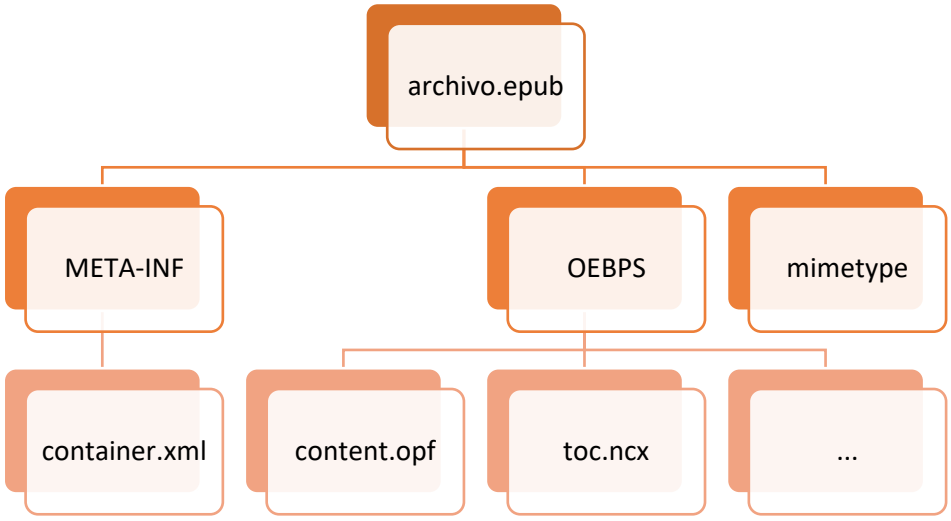


Ilustración 2 - Estructura de un archivo epub

3.2.1 Container.xml

Únicamente hace referencia a la ubicación del archivo content.opf.

3.2.2 Content.opf

Este archivo está dividido en tres partes:

3.2.2.1 Opf:metadata

Contiene información relativa a las características del epub: autor, título…etc.

3.2.2.2 Opf:manifest

Contiene un enlace a todos los archivos que va a utilizar el epub: archivos html, imágenes, css…

3.2.2.3 Opf:spine

Nos dice que la estructura la va a contener el archivo ncx y lista todas las secciones que ese archivo ncx tendrá.

3.2.3 Toc.ncx

Contiene una jerarquía del índice del libro. Si se quisiese insertar un subcapítulo bastará con que ese punto de navegación esté dentro del punto de navegación padre.

3.3 Tablas de la base de datos que vamos a necesitar

La base de datos fue creada usando MyIsam y por lo tanto no existe integridad referencial como tal a pesar de que las tablas están bien construidas y en la creación, borrado y modificación de los elementos nos aseguramos que esa integridad referencial se mantenga.

La base de datos que tenemos dispone de 24 tablas como se puede ver en la siguiente captura:

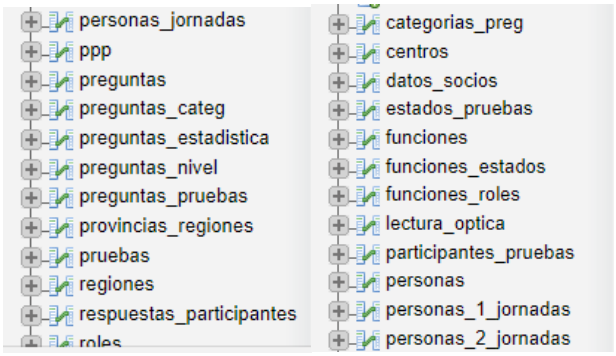


Ilustración 3 - Tablas que contiene el concurso de primavera

Aunque para la realización del libro electrónico únicamente vamos a necesitar las siguientes:

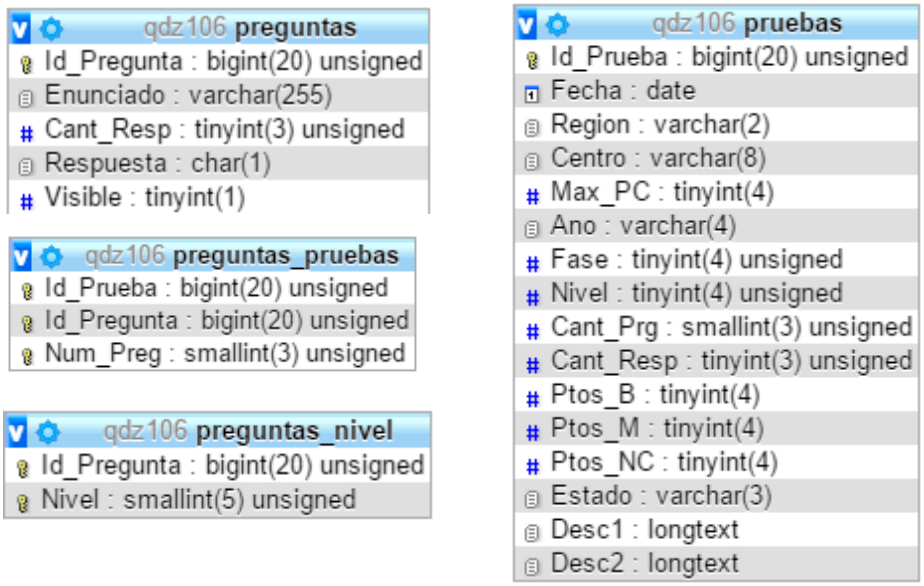


Ilustración 4 - Tablas usadas el concurso de primavera

3.4 Elección del hosting

Se van a barajar únicamente dos alternativas:

Google App Engine: Es una parte del Google Cloud que nos permite alojar y ejecutar aplicaciones. Actualmente existen dos tipos:

- **Google App Standard Environment**: Solo tiene posibilidad de programación con Java, Python, PHP y GO. No Permite accesos a disco usando esto lenguajes, a cambio, los costes son más económicos.
- **Google App Flexible Environment** : Puede implementar cualquier lenguaje de programación mediante contenedores Docker, permitiendo

mayor control sobre la aplicación. Los costes son proporcionales al consumo de CPU.

Jelastic: Es un entorno Cloud PAAS (Plataform As A Service) que permite de forma rápida y sencilla la creación de entornos de trabajo que soportan diferentes lenguajes y tipos de base de datos.

4 DISEÑO

4.1 Arquitectura

Se va utilizar el Patrón Servicio-Repositorio. Este patrón nos permite desacoplar las clases al máximo de modo que cambiando una sola línea de código los datos pueden ser recogidos desde memoria en vez de desde la base de datos. Este desacoplamiento permitirá si fuera el caso poder trabajar en las interfaces de modo autónomo al del desarrollo de los distintos componentes.

Los tres elementos importantes en dicho diagrama son:

- **Modelo**: Van a ser las clases de persistencia que me van a permitir leer/escribir datos de la base de datos usando JPA.
- **Repositorio**: Son los almacenes de información. El repositorio más habitual es la base de datos pero se pueden crear repositorios en memoria por ejemplo.
- **Servicio**: Es la clase que nos permite acceder a los repositorios.

Hay un concepto muy relacionado con estos tres términos que es **Inyección de dependencias** que es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase la que cree el objeto.

También vamos a utilizar la arquitectura **Modelo-Vista-Controlador** que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario.

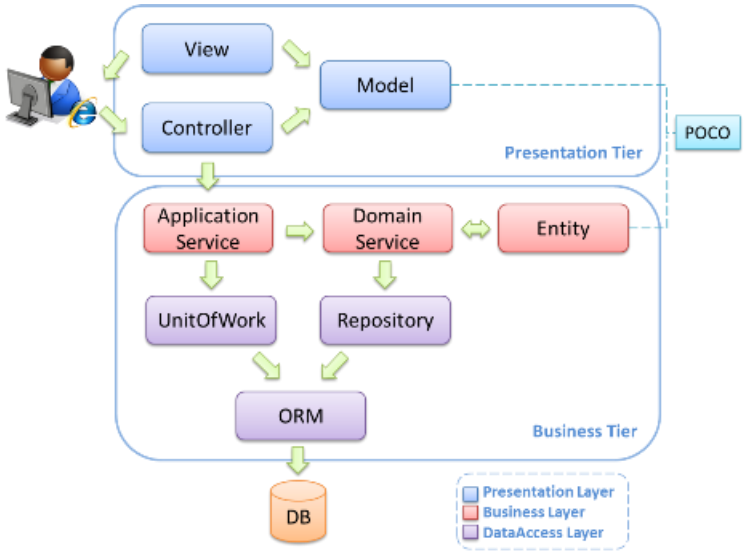


Ilustración 5 – Esquema del patrón servicio-repositorio

4.2 Clases de persistencia en java

Las clases que van a implementar el modelo de la base de datos seleccionado en el apartado *3.3* son las siguientes:

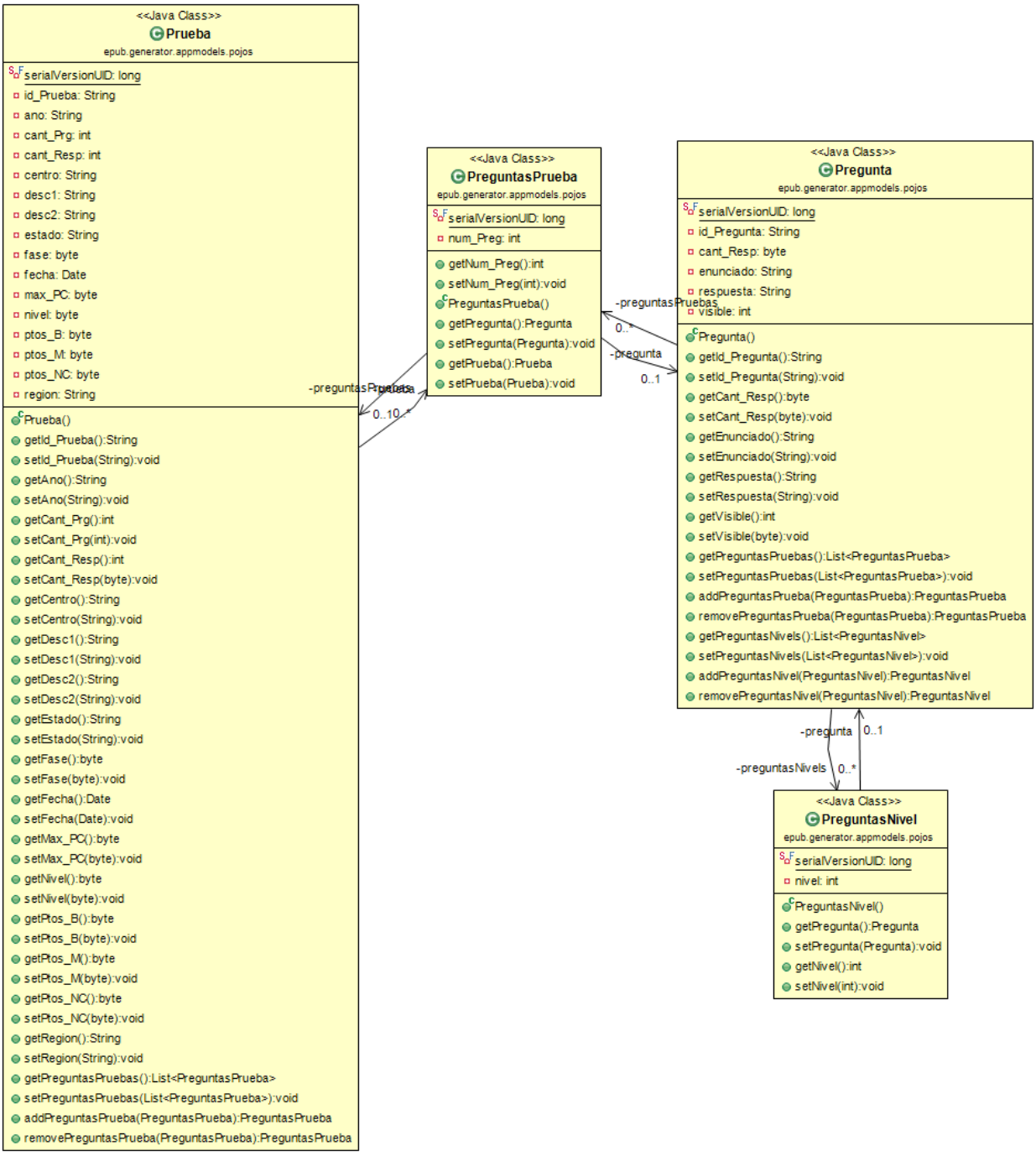


Ilustración 6 - Diagrama de las clases de persistencia

En un futuro va a ser necesario listar esas clases en el archivo persistence.xml para que el gestor de persistencia pueda trabajar sobre ellas:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<persistence>
      <persistence-unit>
      <class>epub.generator.appmodels.pojos.Pregunta</class>
      <class>epub.generator.appmodels.pojos.PreguntasNivel</class>
      <class>epub.generator.appmodels.pojos.PreguntasPrueba</class>
      <class>epub.generator.appmodels.pojos.Prueba</class>
      </persistence-unit>
</persistence>
```

4.3 Diseño de los enlaces entre las páginas del libro electrónico

Debido a que cada pregunta contiene ya un identificador, se va a optar porque las soluciones y los resultados vayan en páginas separadas y cada pregunta contenga un enlace a esa solución haciendo uso de dicho identificador. Por ejemplo si tenemos la prueba Edición I (1999) Fase 1 Nivel1.html, accederemos a la pregunta 15 utilizando el hiperenlace:

Edición I (1999) Fase 1 Nivel1.html#15

Y podremos acceder a su solución en el hiperenlace:

SolEdición I (1999) Fase 1 Nivel1.html#15

4.4 Tipos de pruebas a realizar:

Para asegurarnos el correcto funcionamiento de la aplicación y de los ficheros que la conforman vamos a realizar distintos tipos de pruebas a la aplicación:

- ***Tests Unitarios***: Me permitirán comprobar que todo el código fuente generado funciona correctamente. Concretamente se deberá comprobar:
    - Que el gestor de plantillas funciona correctamente
    - Que las clases de persistencia se leen/graban correctamente
        - Que todas las namedQueries generadas funcionan correctamente
    - Que se genera archivos epub de acuerdo a la información de la base de datos
- ***Tests Funcionales***: Me permitirán comprobar que la funcionalidad del libro electrónico es la adecuada. Las cosas que se deben comprobar son:
    - Que no exista ningún hiperenlace roto dentro del libro.
    - Que todos los archivos que se utilizan están incluidos en el archivo *Content.opf*.
    - Que para cada prueba que exista hay dos secciones (una para los enunciados y una para las respuestas) y cada pregunta de los

enunciados tiene un hiperenlace a la respuesta correspondiente y viceversa

- **Tests de Aceptación**: Un robot irá leyendo las distintas secciones del libro en resolución 600 x 800 y se comprobará las siguientes cosas como si fuera un humano el que lo hace. Si algo no cumpliera estas dos condiciones se generará una captura automática de los errores que se han ocasionado.
    - Que las respuestas no sobrepasen el ancho máximo de 600 pixeles
    - Que las imágenes escalares (png, jpg y gif) no se hayan reducido más de un 50% respecto a su tamaño original, lo que nos asegurará que se visualiza correctamente.

4.5 Diseño de las interfaces

4.5.1 Diseño de la página web

La aplicación de cara al usuario, consistirá únicamente en una página web similar a la siguiente ilustración donde podamos elegir entre los distintos niveles y nos aparezcan las pruebas pertenecientes a cada nivel para ser generadas:

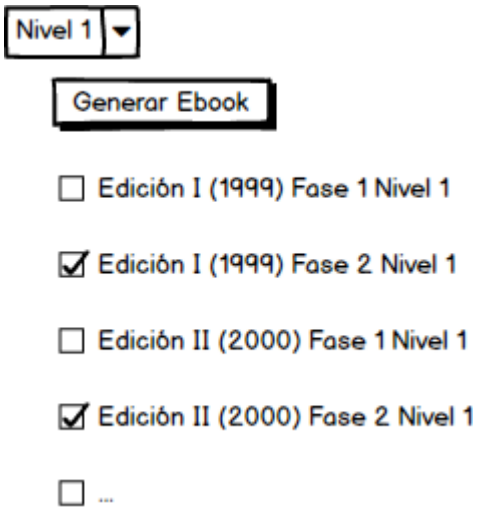


Ilustración 7 - Prototipo de interfaz de la aplicación web

4.5.2 Diseño del ebook

El Ebook debe contar con las siguientes secciones:

- Portada
- Índice de contenidos
- Pruebas: Donde cada prueba debe estar contenida en el índice.

- Soluciones: Cada solución también debe estar contenida en el índice.

4.5.2.1 Portada

El ebook va a contar con una portada que integre las siguientes dos imágenes aunque de momento no sabemos cómo distribuirlas:

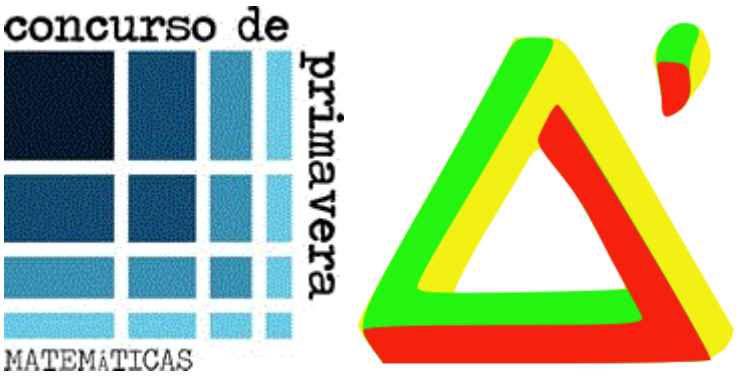


Ilustración 8 - Imágenes de la portada del Ebook

Para la realización de dicha portada ha sido necesaria la conversión del logo de Aprima de formato escalar a formato vectorial, lo que nos permite un re-escalado de la misma sin perder calidad de imagen:

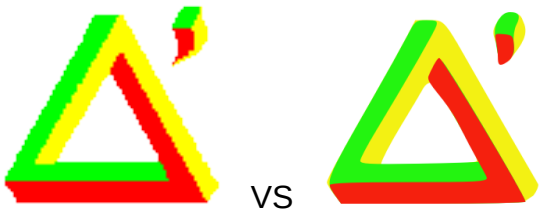


Ilustración 9 - Logo de Aprima en formatos escalar y vectorial

4.5.2.2 Índice

El índice debe contener las distintas secciones de problemas ordenadas cronológicamente y tras éstas sus pertinentes correcciones:

Ilustración 10 - Prototipo del índice del Ebook

4.5.2.3 Pruebas

Va a contener una descripción de la prueba, el logo de Aprima y cada una de las preguntas con un enlace a su respuesta, pero siempre sin mostrar la respuesta directamente en la pregunta:

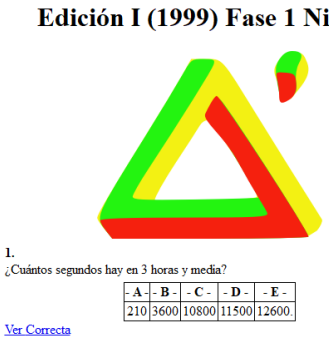


Ilustración 11 - Prototipo de interfaz de una prueba

4.5.2.4 Soluciones

Van a contener una descripción de la prueba, el logo de Aprima y el número de la pregunta junto con su respuesta correcta con ambos datos separados por un guion:

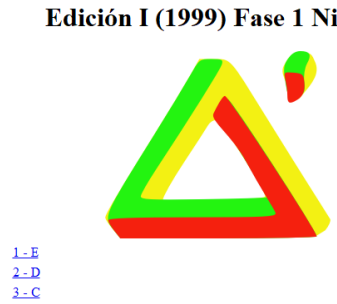


Ilustración 12 - Prototipo de interfaz de la solución a una prueba

5 IMPLEMENTACIÓN

5.1 Implementación de las clases de persistencia

Las clases de persistencia se han generado automáticamente mediante las JPA-Tools, lo que nos va a permitir mediante JPA la recuperación-grabado de la información utilizando dichas clases. A pesar de que las relaciones preguntas-preguntas_pruebas y preguntas-preguntas_nivel son teóricamente 1…1 se han definido como 1..* debido a que no hay en la base de datos ningún sistema de integridad referencial que e impida que así sea.

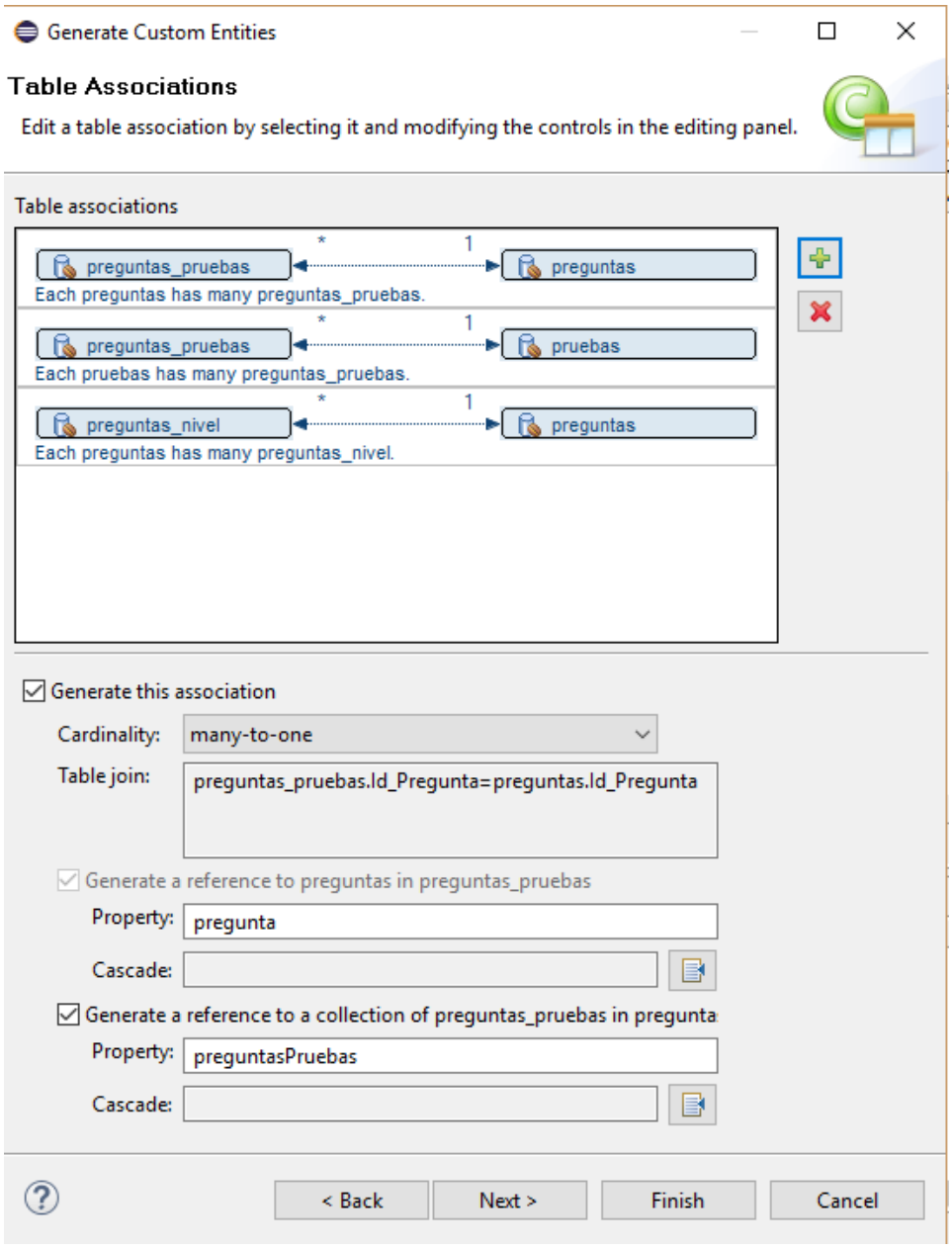


Ilustración 13 - Generación de las clases de Persistencia a través de las JPA-Tools

5.2 Implementación de los servicios-repositorios-modelos

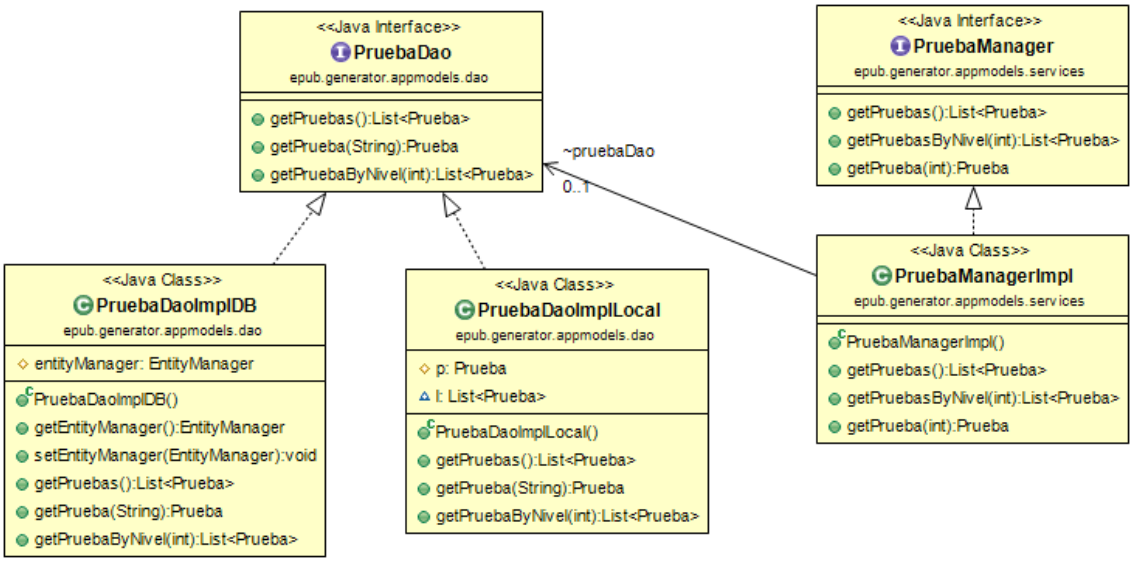


Ilustración 14 - Implementación de las clases usando el patrón servicio-repositorio

Esta sección está directamente relacionada con el apartado *4.1* que se corresponde con el patrón modelo-repositorio.

En dicho patrón vamos a contar con:

- Servicio: es la clase que suministra los datos. En nuestro caso va a ser PruebaManagerImpl, pero le hemos construido por encima una interfaz denominada PruebaManager de modo que podamos generalizar el uso de dicho servicio.
- Repositorio: va a consistir en el almacén de la información. Como en el caso anterior hemos creado una interfaz que también nos permita generalizar el uso de uno u otro almacén de información. En nuestro caso hemos creado dos almacenes de información:
    - PruebaDaoImplDB: que recupera los datos desde la base de datos.
    - PruebaDaoImplLocal: que recupera los datos desde la memoria del ordenador directamente.

La creación de estos dos repositorios es muy ventajosa, ya que por ejemplo, me permite poder probar las interfaces de usuario usando como repositorio PruebaDaoImplLocal sin necesidad de haber creado la base de datos que me permita alojar dichos datos. También me va a permitir con un pequeño cambio de código poder observar si los errores que se producen en la aplicación pueden ser debidos al modelo de base de datos

o a la propia conexión de la base de datos, o bien son errores de programación.

Cada uno de esos elementos se define mediante las consecuentes anotaciones, estas son:

@Service: Servicio que me suministrará datos.

@Repository: Almacén de Información.

@Entity: Clase de Persistencia que mediante ésta anotación podrá ser grabada/leída de la base de datos.

5.3 Inyección de las dependencias

La inyección de dependencias nos va a permitir desacoplar al máximo el código fuente. El ejemplo más latente de esto es la obtención del entityManager que nos va a permitir cambiar la configuración de la base de datos de modo sencillo para todo el proyecto.

5.3.1.1 Inyección de dependencias desde un archivo xml

Esto se hace a través de varios pasos. El primero es definir en el dispatcher los beans que cubran la funcionalidad deseada:

```
<bean id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSour
ce">
            <property name="driverClassName"
value="com.mysql.jdbc.Driver" />
            <property name="url"
value="jdbc:mysql://databaseurl:3306/databasename" />
            <property name="username" value="databaseuser" />
            <property name="password" value="databasepwd" />
      </bean>

<bean id="entityManagerFactory"
      class="org.springframework.orm.jpa.LocalContainerEntityManagerFa
ctoryBean">
            <property name="persistenceXmlLocation"
value="classpath:META-INF/persistence.xml" />
            <property name="persistenceUnitName"
value="989812893192389" />
            <property name="dataSource" ref="dataSource" />
            <property name="jpaPropertyMap">
                  <props>
                        <prop key="eclipselink.weaving">false</prop>
                  </props>
            </property>
      </bean>
```

5.3.2 Inyección de dependencias desde una clase Java

Se puede realizar la misma inyección de dependencias pero mediante el uso de clases java que irán anotadas con la etiqueta @Bean

```
@Bean
public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
	EclipseLinkJpaVendorAdapter vendorAdapter = new
EclipseLinkJpaVendorAdapter();
	vendorAdapter.setGenerateDdl(true);
	vendorAdapter.setShowSql(false);

	LocalContainerEntityManagerFactoryBean factory = new
LocalContainerEntityManagerFactoryBean();
	factory.setJpaVendorAdapter(vendorAdapter);
	factory.setPackagesToScan("epub.generator");
	factory.setDataSource(dataSource());
	factory.setJpaProperties(jpaProperties());
	try {
		factory.setLoadTimeWeaver(loadTimeWeaver());
	} catch (Throwable e) {
		e.printStackTrace();
	}
	return factory;
  }
```

5.3.3 La anotación @PersistenceContext

Lo siguiente que debemos hacer es definir un atributo en la clase que deseemos y establecerle los getters y los setters para que la inyección de dependencias pueda ocurrir:

```
@Repository
public class PruebaDaoImplDB implements PruebaDao {

	@PersistenceContext
	protected EntityManager entityManager;

	public EntityManager getEntityManager() {
		return entityManager;
	}

	public void setEntityManager(EntityManager entityManager) {
		this.entityManager = entityManager;
	}
```

Si quisiéramos inyectar otro atributo en vez del Gestor de Persistencia debemos usar la anotación @Autowired.

5.4 Creación de las páginas del libro electrónico con thymeleaf

La ventaja que me aporta thymeleaf es que me permite la creación de los prototipos de interfaz y a la vez me permite la creación final de las mismas todo ello en html, tal y como ilustro:

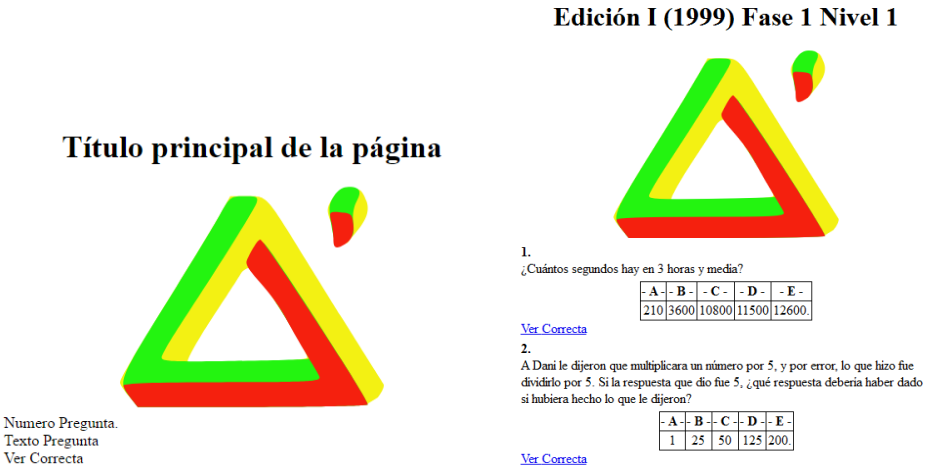


Ilustración 15 - Prototipo de interfaz y resultado final usando thymeleaf

Código de la plantilla:

```
<body>
      <center>
      <h1 th:text="${descripcion}">Título principal de la página</h1>
      </center>
      <center>
            <img src="../primavera/graficosp/primavera.svg" />
      </center>
      <table>
            <tr th:each="preg:${preguntas}"><td>
      <div th:utext="'<b>'+${preg.id}+'.</b>'">Numero Pregunta.</div>
      <div th:id="${preg.id}" th:utext="${preg.pregunta}">Texto
                              Pregunta</div> <a
th:href="'Sol'+${descripcion}+'.html#'+${preg.id}">Ver
                              Correcta</a>
      </td></tr>
      </table>
  </body>
```

Para poder ejecutar thymeleaf es necesario crear una instancia de la clase TemplateEngine y tal y como se describió en la sección *4.1* la clase TemplateEngineFactory se ha programado como un repositorio. Lo más reseñable de la clase es que establezco las propiedades que van a tener las plantillas de los archivos. Lo más reseñable es que establezco el tipo de plantilla que vamos a recibir, la codificación, la ubicación de las plantillas

```
public final static String TEMPLATEMODE =  "HTML";
public final static String TEMPLATEEXTENSION=  ".html";
public final static String TEMPLATEPATH=  "./templates/";
public final static String OUTPUTENCODING = "UTF-8";
```

```
      //Datos del servidor d


private static void initializeTemplateEngine(boolean cacheable, Long
timeToLive)
    {
      ClassLoaderTemplateResolver templateResolver = new
ClassLoaderTemplateResolver();
     templateResolver.setTemplateMode(Constantes.TEMPLATEMODE);
     templateResolver.setSuffix(Constantes.TEMPLATEEXTENSION);
     templateResolver.setCacheable(cacheable);
     templateResolver.setCacheTTLMs(timeToLive);
     templateResolver.setPrefix(Constantes.TEMPLATEPATH);
     templateResolver.setCharacterEncoding(Constantes.OUTPUTENCODING);

     templateEngine = new TemplateEngine();
     templateEngine.setTemplateResolver(templateResolver);
}
```

5.5 Problemas encontrados

5.5.1 Problemas en la implementación

5.5.1.1 Codificación de los datos en la Base de Datos

Debido a que en la base de datos los datos se almacenan con codificación ISO 8859-1 ha sido necesario crear un conversor en las clases de persistencia que me permita leer/grabar dicho campo de modo adecuado.

	Id_Prueba	Fecha	Region	Centro	Max_PC	Ano	Fase	Nivel	Cant_Prg	Cant_Resp	Ptos_B	Ptos_M	Ptos_NC	Estado	Desc1	Desc2
	97	2011-02-23	26	*NULL*	4	2011	1	1	25	5	5	0	2	800	EdiciÃ³n XIII (2011) Fase 1 Nivel 1	*NULL*

Ilustración 16 - Tupla de la tabla prueba

Este es el código fuente del converso. Utiliza dos métodos que vienen heredados de la clase AttributeConverter:

- **convertToDatabaseColumn**: Convierte el valor del atributo cuando este va a ser grabado en la base de datos.
- **convertToEntityAttribute**: Convierte el valor del campo de la base de datos cuando es leído y se almacena como atributo del objeto.

```
@Converter
public class ChangeEncodingConverter
implements AttributeConverter<String, String> {

      public String convertToDatabaseColumn(String arg0) {
            String decoded = arg0;
            try {
                  decoded = new
String(arg0.getBytes(Constantes.OUTPUTENCODING),Constantes.INPUTDBENCO
DING);
            } catch (UnsupportedEncodingException e) {
```

```
                    // TODO Auto-generated catch block
                    e.printStackTrace();
              }
              return decoded;
        }

        public String convertToEntityAttribute(String arg0) {
              String decoded = arg0;
              try {
                    decoded = new
String(arg0.getBytes(Constantes.INPUTDBENCODING));
              } catch (UnsupportedEncodingException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
              }
              return decoded;
        }
   }
```

Y la anotación correspondiente para llamar a dicha clase desde Prueba.java:

```
@Lob
@Convert(converter=ChangeEncodingConverter.class)
private String desc1;
```

5.5.1.2 Cambio de las clases con claves externas

Tal y como las JPA-Tools creaban las clases de persistencia era imposible utilizarlas y ha habido que cambiar la forma en la que las claves son mapeadas en clases de persistencia. Por ejemplo para preguntas_nivel:

1. En la clase PreguntasNivel ha habido que decir qué clase contiene la clave que es PreguntasNivelPK: `IdClass(PreguntasNivelPK.class)`
2. Se han cambiado los tipos de los atributos de PreguntasNivel:

   ```
   @Id
   private int nivel;

   @Id
   @ManyToOne
   @JoinColumn(name="id_Pregunta")
   private Pregunta pregunta;
   ```

3. Se han cambiado los tipos en PreguntasNivelPK y nivel ha pasado a ser un int y pregunta ha pasado a ser un String

5.5.1.3 Código de epublib que no funciona

Se ha detectado que la librería epublib a la hora de extraer los archivos que contiene el archivo .epub no lo hace correctamente. Se ha observado que el error proviene del método readEpub de la clase EpubReader.

Se ha tratado de solucionar el problema extendiendo dicha clase y reescribiendo el método pero como la visibilidad del método es protected ha habido que reescribir gran parte del código.

```
public class EpubReaderExtend extends EpubReader {


	public Book readEpub(String filename) throws IOException {}

	private Resource processNcxResource(Resource packageResource,
Book book) {}

	private Resource processPackageResource(String
packageResourceHref, Book book, Resources resources) {}

	private Book postProcessBook(Book book) {}

	private Book readEpub(ZipFile in, String encoding) throws
IOException {}

	private String getPackageResourceHref(Resources resources) {}

	private void handleMimeType(Book result, Resources resources) {}

	private Resources readResourcesExt(ZipFile in, String
encoding){}

   }
```

5.6 Cambios que ha sido necesario introducir en el código

5.6.1 Diferencia entre ejecución local vs entorno web

La generación de los epub se realiza utilizando rutas relativas y al realizarse el empaquetado en un archivo war de los contenidos esas rutas relativas son distintas de las que contienen los proyectos empaquetados.

Por suerte todas esas rutas habían sido definidas en un archivo de constantes y para que la generación funcione únicamente ha habido que editar éste contenido:

```
ServletContext context = request.getServletContext();
String pa = context.getRealPath("/WEB-INF/classes/estilo.css");
File file = new File(pa).getParentFile();
Constantes.CARPETAARCHIVOS = file.getAbsolutePath()+File.separator;
Constantes.CARPETAIMAGENES = Constantes.CARPETAARCHIVOS +"graficosp/";
Constantes.CARPETAENUNCIADOS=Constantes.CARPETAARCHIVOS + "pruebasb/";
```

5.7 Alojamiento de la aplicación

Esta es la parte que más quebraderos de cabeza ha dado durante el proyecto, ya que por comodidad y por estar ampliamente extendido se ha decidido utilizar

los servicios de Hosting de Google Cloud por la sencilla razón que nos da 300 €uros de prueba para poder gastar a lo largo de un año.

Ilustración 17 - Infraestructura Google Cloud Platform

La pega que nos presentan dichos servicios es que frente a un despliegue normal de una aplicación utilizando un archivo con extensión war, Google App es muchísimo más rígido y restringe entre otras cosas las librerías y clases que puedes utilizar y los orígenes de los datos. Las dificultades que nos hemos encontrado han sido las siguientes:

1. Hay que crear un proyecto nuevo con un arquetipo nuevo específico de Google App Engine y convertir el proyecto web antiguo para que se adecue a esta estructura.
2. Hay que compilar el proyecto con el JDK 1.7 y todas las librerías referenciadas en el proyecto tienen que haber sido compiladas con la versión 1.7, por lo que ha habido que bajar la versión de numerosas librerías utilizadas durante el proyecto
3. No puede haber escritura de ficheros en el código fuente, por lo que he tenido que cambiar parte de la programación de las funciones para convertir esos archivos que se generaban a streams que van pasando entre las distintas funciones del programa.
4. Google App Engine está limitado a 10000 archivos por proyecto y como nuestros archivos html y de imagen iban incluidos en la compilación sobrepasamos ese límite por lo que ha habido que migrar esos archivos estáticos al Cloud Storage.
5. Desde Google App Engine no se puede acceder a bases de datos sql que estén en el exterior, por lo que ha habido que migrar la parte de la base de datos descrita en la sección *3.3* a Google Cloud SQL.

6. Ha habido que modificar el código fuente original puesto que Google limita el uso de ciertas clases Java por motivos de Seguridad, a continuación muestro dos ejemplos que sirven para recorrer carpetas del Google Cloud Storage. Los cambios que ha habido que realizar los describo en la sección 5.7.4.

La ubicación donde Google nos ha alojado la aplicación es la siguiente:

https://epubgenerator-171116.appspot.com/

A continuación describo varios aspectos de los descritos anteriormente un poco más en profundidad.

5.7.1 Subida de archivos al Google Cloud Storage

La subida de los archivos se ha realizado usando la línea de comandos, a continuación muestro el ejemplo de las dos cosas que ha habido que hacer.

Subir los elementos al Google Cloud:

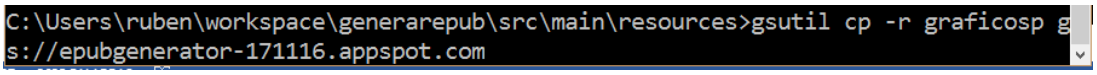


Ilustración 18 - Subida recursiva de archivos a Google Cloud Storage

Darles permisos de lectura para cualquier usuario:

Ilustración 19 - Cambio recursivo de permisos en archivos de Google Cloud Storage

5.7.2 Creación del Archivo json de autenticación

Para poder acceder a los datos del Google Cloud Storage desde Java y descargarlos, hay que hacerlo cargando un archivo con extensión json que nos va a autorizar a realizar dichas operaciones:

```
Storage storage = StorageOptions.builder()
.authCredentials(AuthCredentials.createForJson(new
FileInputStream("cred.json")))
   .build().service();
```

Para crear dicho archivo ha habido que dirigirse a la siguiente web:

https://console.cloud.google.com/apis/credentials

Hemos ido a Credenciales->Crear credenciales -> Clave de cuenta de servicio

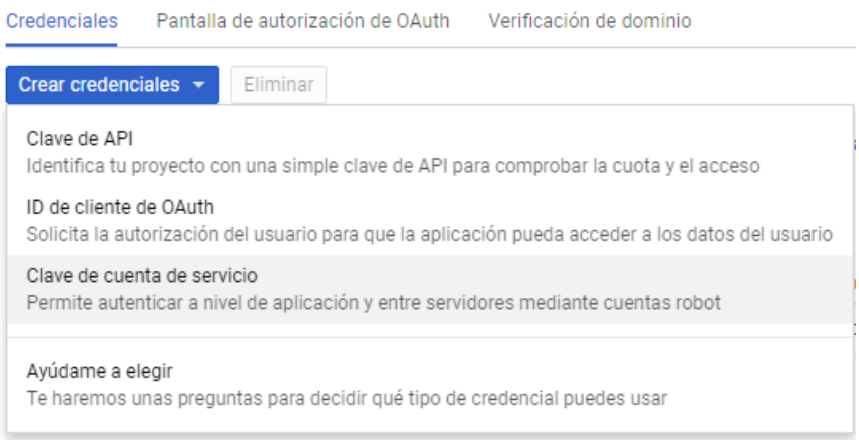


Ilustración 20 - Creación de los credenciales como cuenta de servicio

Seleccionamos Appengine y escogemos Json como formato:

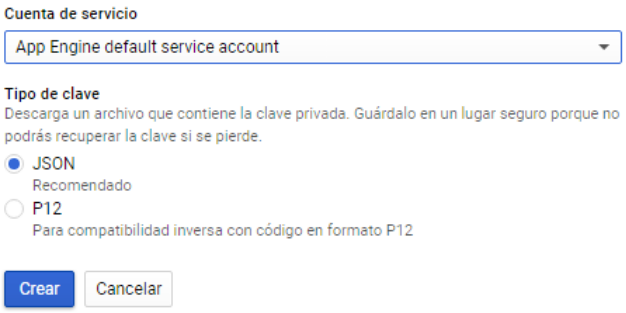


Ilustración 21 - Selección del credencial como archivo JSON

Ahora desde el menú podemos observar la clave y ya habremos descargado el archivo:

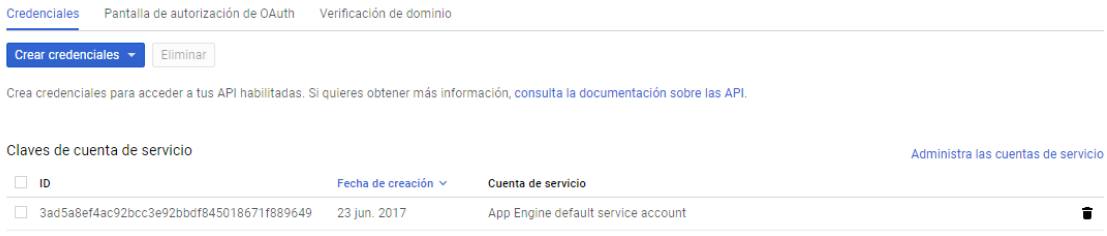


Ilustración 22 - Visualización del credencial generado

5.7.3 Creación de la base de datos en Google Cloud SQL

Lo primero que debemos hacer es ingresar en la página web y crear una instancia nueva:

Ilustración 23 - Creación de una instancia de base de datos

Seleccionamos una instancia MySQL y le damos a siguiente:

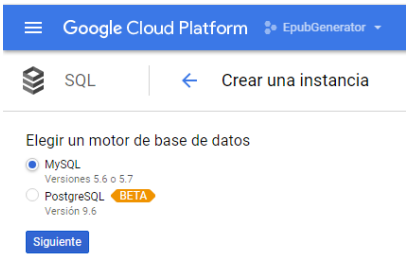


Ilustración 24 - Selección del motor de base de datos como MySQL

Elegimos la segunda generación:

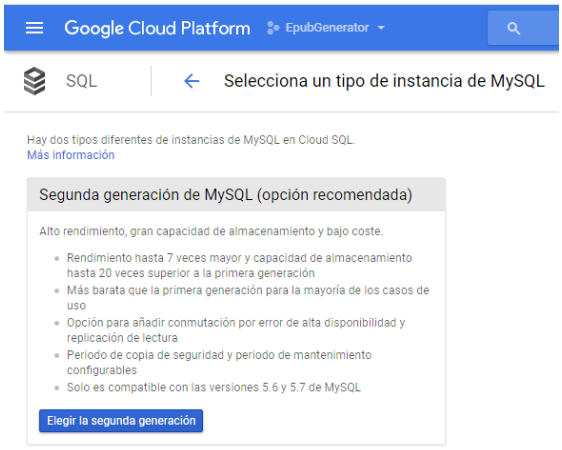


Ilustración 25 - Elección del motor de generación de base de datos

Seleccionamos el nombre de la instancia y la contraseña del usuario root:

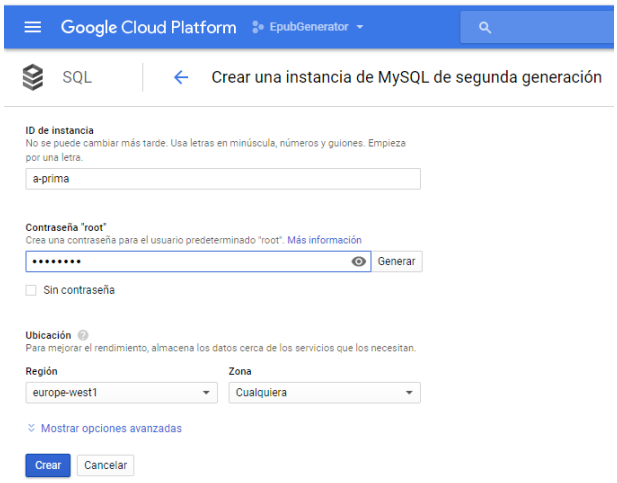


Ilustración 26 - Generación de la contraseña para el usuario root

Le damos a crear y ya tendremos la instancia creada:

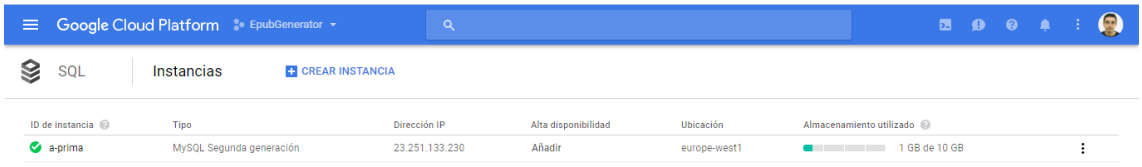


Ilustración 27 - Visualización de la instancia de base de datos

Ahora deberemos subir los archivos .sql que creamos convenientes haciendo click en importar:

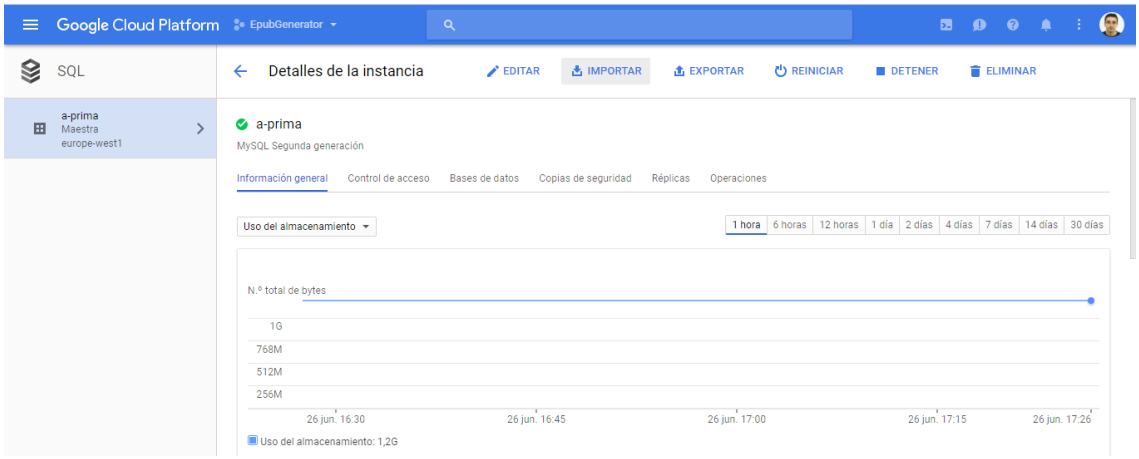


Ilustración 28 - Importar archivos .sql a una instancia de base de datos

Y seleccionamos el archivo .sql que deberá haber sido subido previamente a Google Cloud Storage siguiendo las instrucciones del punto [5.7.1.]

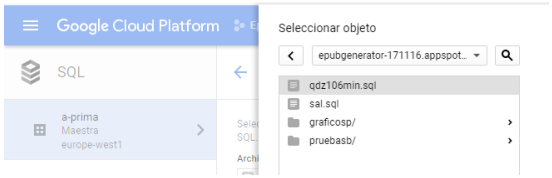


Ilustración 29 - Selección del archivo .sql para ser importado

Lo último que nos quedará es dar permisos a la aplicación para acceder a los servicios de Google Cloud Sql es dar permisos desde la pestaña IAM Administración:

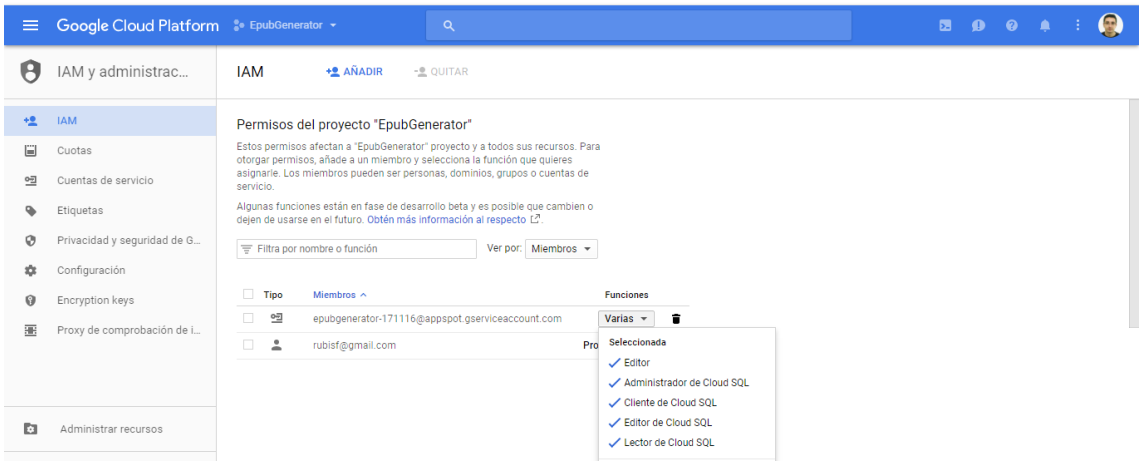


Ilustración 30 - Creación de permisos para el acceso a la base de datos

5.7.4 Modificación del código fuente evitando el paquete java.nio

Google App Engine limita el uso de ciertas clases Java y sucede esto con toda la gama del paquete java.nio, por lo que hemos debido reconvertir nuestro código de descargar archivos y listar carpetas haciendo uso de una librería externa:

5.7.4.1 Lectura de archivos

Versión java.nio:

```
Path path = Paths.get(URI.create("gs://epubgenerator-
171116.appspot.com/pruebasb/p1/p1.htm"));
            try (InputStream input = Files.newInputStream(path))
```

Versión sin java.nio:

```
Blob blob = storage.get("epubgenerator-171116.appspot.com",
"pruebasb/p1/p1.htm");
                  ReadChannel readChannel = blob.reader();
                  FileOutputStream      fileOuputStream      =      new
FileOutputStream("p2.htm");
```

5.7.4.2 Listado de Carpetas

Versión java.nio:

```
Path p = CloudStorageFileSystem.forBucket("epubgenerator-
171116.appspot.com", csc, storage.options())
                        .getPath("graficosp/p1/");
            try (DirectoryStream<Path> stream =
Files.newDirectoryStream(p)) {
                  for (Path path : stream) {
                        System.out.println(path);

                  }
```

Versión sin java.nio:

```
Page<Blob> blobs = storage.list("epubgenerator-171116.appspot.com",
BlobListOption.currentDirectory(),
                        BlobListOption.prefix("graficosp/p1/"));
                  for (Blob blob : blobs.values()) {
                  System.out.println(blob.name());
  }
```

5.7.4.3 Cambio entre versiones de desarrollo en el Google App Engine

Podemos establecer la versión de la aplicación desde el archivo pom.xml y ese valor luego nos sirve para poder cambiar entre versiones, por ejemplo yo creé una versión de prueba inicial que contiene en el nombre el día que la creé y a partir de ahora he decidido numerarlas correlativamente… 1, 2, 3 etc.

La captura muestra esos datos y como el tráfico está desviado a la versión 1:

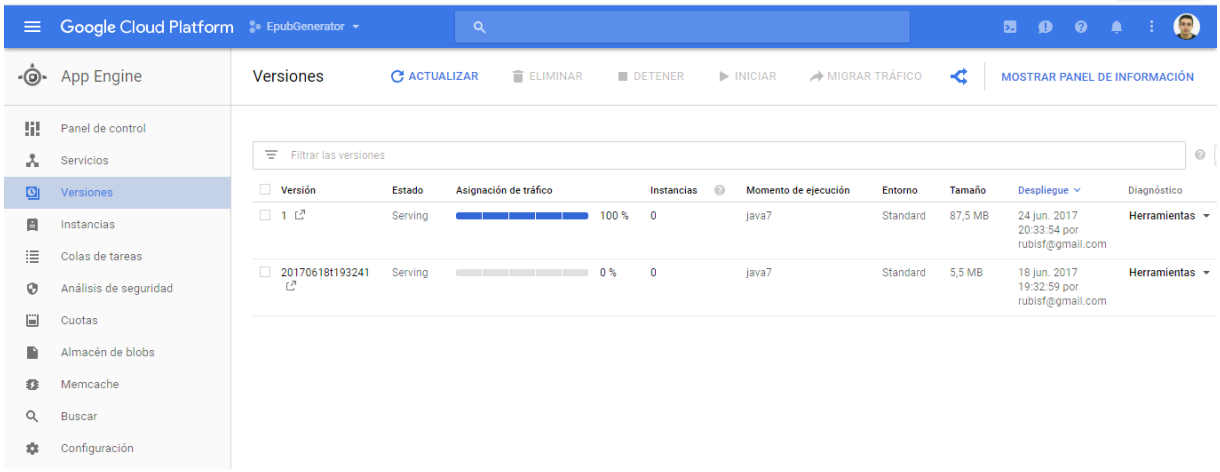


Ilustración 31 - Versiones del programa disponibles

5.7.5 Cambiando el timeout de Google App Engine

Por defecto es imposible cambiar el timeout que Google considera que es el máximo para una aplicación y que fija en 60 segundos.

Lo que si se puede hacer es cambiar el escalado a manual y de éste modo que ese tiempo sea infinito. Esto se hace manipulando el archivo appengine-web.xml y añadiendo las líneas:

```
<instance-class>B8</instance-class>
<manual-scaling>
      <instances>1</instances>
</manual-scaling>
```

Los tipos de servicio disponibles son los siguientes:

Tipo de Instancia	Límite de Memoria	Límite de CPU	Escalados soportados
B1	128 MB	600 MHz	manual, basic
B2	256 MB	1.2 GHz	manual, basic
B4	512 MB	2.4 GHz	manual, basic
B4_1G	1024 MB	2.4 GHz	manual, basic
B8	1024 MB	4.8 GHz	manual, basic
F1	128 MB	600 MHz	automatic
F2	256 MB	1.2 GHz	automatic
F4	512 MB	2.4 GHz	automatic
F4_1G	1024 MB	2.4 GHz	automatic

5.7.6 Precio de los servicios

5.7.6.1 Google App Engine

El costo de alojar una aplicación en Google App Engine depende sobretodo de la potencia del servidor en el que alojemos nuestra aplicación y del número de instancia que creemos (a multiplicar estos dos parámetros). A continuación muestro un listado con los precios para el servidor de Bruselas.

Tipo de instancia	Costo por hora de instancia
B1	$0.05
B2	$0.10
B4	$0.20
B4_1G	$0.30
B8	$0.40
F1	$0.05
F2	$0.10
F4	$0.20
F4_1G	$0.30

Como se puede observar la hora del servidor de tipo B8 quintuplica la del servidor B1. Esto ha ocasionado que en pocos días haya gastado más de 60 dólares en uso de backend y haya tenido que dejar la aplicación apagada a partir del 11 de Julio de 2017 porque me queda menos de la mitad del crédito inicial:

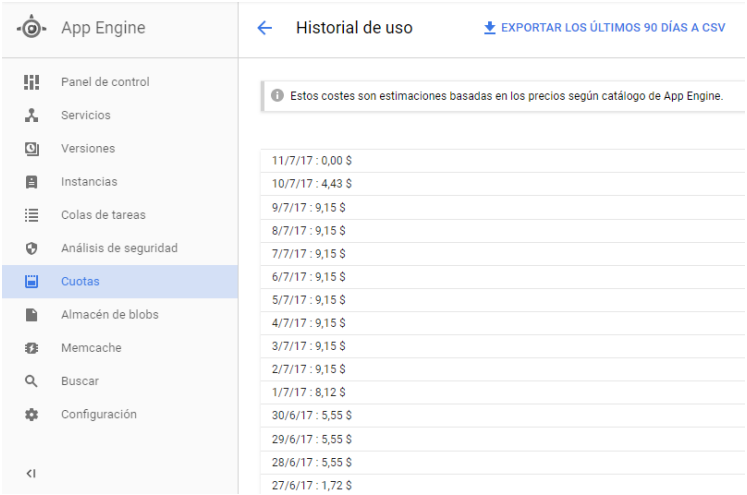


Ilustración 32 - Costo generado por alojar la aplicación en Google App Engine

5.7.6.2 Google Cloud Storage

El costo de alojar datos en el Google Cloud Storage depende del tipo de almacenamiento que deseemos como muestro a continuación.

Tipo de Almacenamiento	Cuando usarlo
Multi-Regional Storage	Datos que son servidos a páginas de todo el mundo tales como el contenido de páginas web, vídeos para ser reproducido en streaming o datos de juegos y aplicaciones móviles.
Regional Storage	Datos accedidos desde la misma región o desde tu aplicación de Google App Engine.
Nearline Storage	Datos que no esperas que sean solicitados con frecuencia (ej. Una vez al mes). Esta configuración es ideal para backups.
Coldline Storage	Datos que no esperas que se acceda a ellos casi nunca (ej. Una vez al año). Habitualmente es usado para guardar backups o datos que quieres conservar pero no esperas que se acceda a ellos.

El precio es bastante barato como muestro en la tabla a continuación:

Multi-Regional Storage (GB por Mes)	Regional Storage (GB por Mes)	Nearline Storage (GB por Mes)	Coldline Storage (GB por Mes)
$0.026	$0.02	$0.01	$0.007

Esto quiere decir que por aproximadamente medio dólar al mes podemos transferir un giga de datos mensual por cada uno de los tipos de servicios descritos:

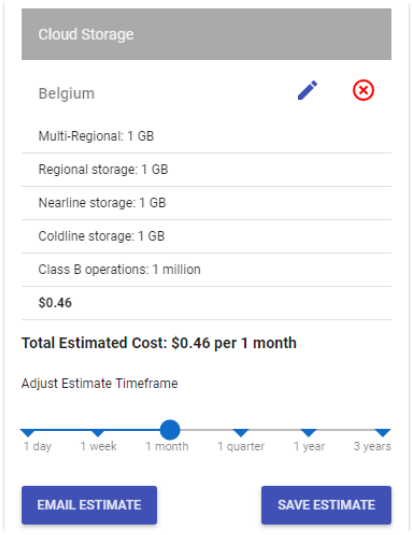


Ilustración 33 - Costo Google Cloud Storage

5.7.6.3 Google Cloud SQL

Me voy a centrar en el servicio que yo he contratado que es la Instancia MySQL de segunda generación. El costo lo muestro a continuación:

Tipo de Máquina	CPUs Virtuales	RAM (GB)	Máxima capacidad de almacenamiento	Precio ($/hora)	Precio sostenido de uso ($/hora)
db-f1-micro	Shared	0.6	3,062 GB	$0.0150	$0.0105
db-g1-small	Shared	1.7	3,062 GB	$0.0500	$0.0350
db-n1-standard-1	1	3.75	10,230 GB	$0.0965	$0.0676
db-n1-standard-2	2	7.5	10,230 GB	$0.1930	$0.1351
db-n1-standard-4	4	15	10,230 GB	$0.3860	$0.2702
db-n1-standard-8	8	30	10,230 GB	$0.7720	$0.5404
db-n1-standard-16	16	60	10,230 GB	$1.5445	$1.0812
db-n1-standard-32	32	120	10,230 GB	$3.0885	$2.1620
db-n1-highmem-2	2	13	10,230 GB	$0.2515	$0.1761
db-n1-highmem-4	4	26	10,230 GB	$0.5030	$0.3521
db-n1-highmem-8	8	52	10,230 GB	$1.0060	$0.7042

db-n1-highmem-16	16	104	10,230 GB	$2.0120	$1.4084
db-n1-highmem-32	32	208	10,230 GB	$4.0240	$2.8168

Esto quiere decir que la opción más barata nos va a costar aproximadamente 35 céntimos al mes aunque yo he seleccionado una máquina algo mejor:

Ilustración 34 - Tipo de Instancia elegida Google Cloud SQL

5.8 Maven

Maven es una herramienta software que me permite la gestión y construcción de proyectos Java de modo fácil a través del concepto de artefacto. Un artefacto es un archivo denominado pom.xml que incluye toda la información necesaria relativa a un proyecto. Entre esos datos constan:

- Nombre del proyecto.
- Grupo al que pertenece.
- Versión del programa.
- Dependencias que necesita satisfacer.

Es importante saber que la gestión de dependencias debe incluir la versión de la dependencia que se necesita y que a su vez, esta librería puede depender de otras y generar una estructura de dependencias compleja. Para nuestro proyecto el gráfico de dependencias utilizadas es el que corresponde a la siguiente ilustración:

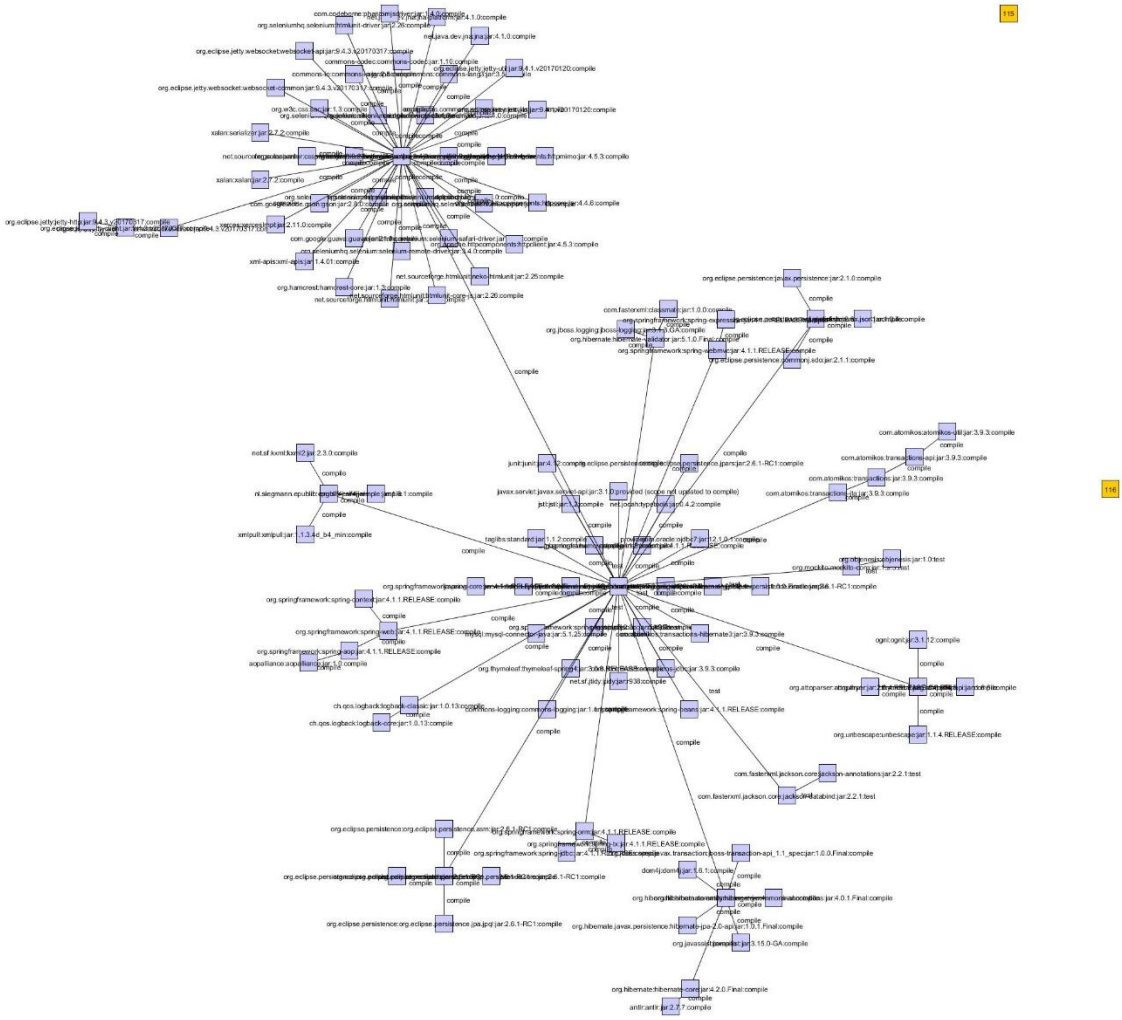


Ilustración 35 - Gráfico de dependencias del proyecto

Una vez establecida una versión de una librería que funciona adecuadamente con nuestro código puede resultar complejo saber cuándo esas librerías se actualizan para de este modo adecuar nuestro programa a las últimas versiones pero esto se puede solucionar mediante la librería [versions-maven-plugin]. Esta librería me permite automáticamente detectar qué versión es la última que se ha publicado y mostrarme un mensaje de advertencia si la versión que estoy utilizando es anterior a la versión existente en los servidores. Si así lo deseamos podemos realizar una actualización de todas las dependencias del proyecto utilizando el comando: **mvn versions:use-latest-versions**

Adicionalmente se pueden realizar una gran cantidad de acciones, entre las que destacamos las siguientes:

- **mvn clean**: Limpia el directorio donde se generan los resultados de la compilación.
- **mvn appengine:update**: Sube el proyecto al google app engine.

- **mvn install**: Lanza los tests y genera los archivos de compilación.
- **mvn appengine:devserver_start**: Arranca un servidor jetty que aloja la aplicación.
- **mvn appengine:devserver_stop:** Para el servidor arrancado en el punto anterior.
- **mvn dependency:tree -DoutputType=graphml**: Genera gráfico de dependencias.
- **mvn test** ejecuta las pruebas unitarias.

Algunos de esos comandos coinciden con los ciclos de vida de construcción por defecto que tiene Maven. Esos ciclos de vida se encargan de construir nuestro proyecto y para llegar a una fase posterior del ciclo de vida será necesario haber pasado por las fases anteriores. Los ciclos de vida básicos son:

Nombre de la fase del ciclo de vida	Descripción
validate	Valida el proyecto si es correcto y si toda la información necesaria está disponible
compile	Compila el código del proyecto
test	Ejecuta las pruebas utilizando un Framework apropiado para los test unitarios (por ejemplo “JUnit”). Estas pruebas no deben requerir que el código esté empaquetado (packaged) o desplegado (deployed)
package	Empaquetar el código compilado en un formato distribuible, como en un JAR, WAR, EAR, EJB, POM, MAVEN-PLUGIN
integration-test	Procesar (process) y desplegar (deploy) el paquete si fuera necesario en un entorno

	donde se puedan ejecutar pruebas de integración
verify	Ejecutar todas las comprobaciones para verificar la validez del paquete y que cumpla con los criterios de calidad
install	Instala el paquete en un repositorio local, para poder utilizarse como dependencia en otros proyectos locales
deploy	Realizar en un entorno de integración o de producción (release), copia el proyecto final al repositorio remoto para compartirlo con otros desarrolladores y proyectos

Las fases de Maven se pueden dividir en sub-fases como muestro en la imagen que se visualiza a continuación:

Ilustración 36 - Fases del ciclo de vida de maven

Existe una correspondencia entre esas fases y el arquetipo que es necesario invocar en el archivo pom.xml para trabajar con cada una de ellas:

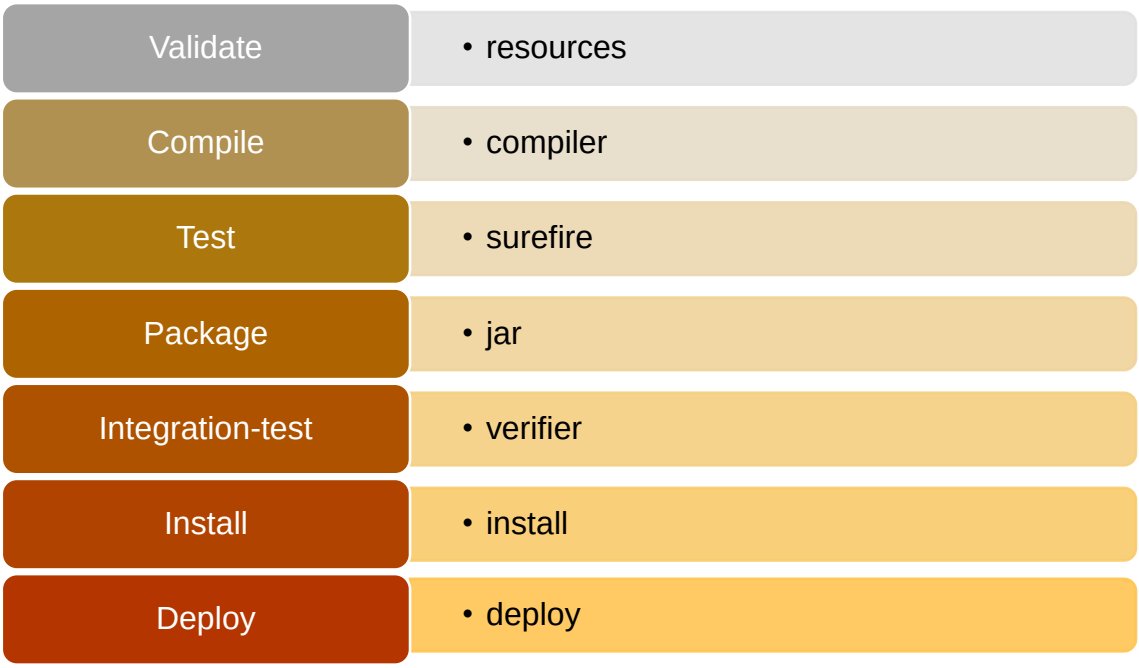


Ilustración 37 - Equivalencia entre las fases de maven y los plugins que hay que usar

6 PRUEBAS

Tal y como se describió en la sección *4.4* se van a realizar varios tipos de pruebas a la aplicación:

- ***Tests Unitarios***: Me permitirán comprobar que todo el código fuente generado funciona correctamente.
- ***Tests Funcionales***: Me permitirán comprobar que la funcionalidad del libro electrónico es la adecuada.
- **Tests de Aceptación**: Un robot irá leyendo las distintas secciones del libro en resolución 600 x 800 y se comprobarán las secciones del libro como si fuera un humano el que lo hace.

6.1 Implementación de las clases de testing

6.1.1 Utilización de las anotaciones

Las pruebas unitarias deben ser independientes unas de otras y como tal la máquina virtual las ejecuta en el orden que le da la gana. Para poder trabajar de un modo adecuado sobre las pruebas ha sido necesario usar los siguientes atributos:

1. @BeforeClass = Se utiliza una única vez previo a la ejecución de la clase. En el método anotado de este modo, se crean archivos que sean muy costosos en tiempo, por ejemplo los epub.
2. @Before = Se utiliza múltiples veces por clase y lo hace previo a la ejecución de cada método de la clase. En el método anotado de éste modo se inicializa la transacción y se crean en la base de datos los datos necesarios para poder realizar los tests.
3. @After = Se utiliza múltiples veces por clase y lo hace de modo posterior a la ejecución de cada método de la clase. En el método anotado de éste modo se hacen los commit y se borran todos los datos generados para dicho test.
4. @AfterClass = Se utiliza una única vez posterior a la ejecución de los tests. Se utiliza para eliminar todos los archivos y carpetas generados durante los tests.

6.1.2 Implementación del testing de los controladores

Los controladores en spring-mvc tienen unas características que los hacen difíciles de poder comprobar su funcionamiento. Para ello spring dispone de sus propias librerías que nos permiten hacer las comprobaciones como muestro a continuación. El siguiente trozo de código nos permite comprobar que al llamar a la ruta `/nivel/9` vamos a obtener lo mismo que a través del servicio, y nos vamos a asegurar que este controlador llama a la vista `nivel.jsp`

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes= {WebAppContext.class})
@WebAppConfiguration
public class ControllerTest {

	private static MockMvc mockMvc;

	@Autowired
	private PruebaManager pruebaManager;

	// Add WebApplicationContext field here


	@Before
	public void iniciar(){
		MockitoAnnotations.initMocks(this);
		SampleController sc = new SampleController();
		sc.setPruebaManager(pruebaManager);

		 mockMvc = MockMvcBuilders.standaloneSetup(sc)

.setHandlerExceptionResolvers(WebAppContext.exceptionResolver())
				.setViewResolvers(WebAppContext.viewResolver())
				.build();
	}

	@Test
	public void
findAll_ShouldAddTodoEntriesToModelAndRenderTodoListView() throws
Exception {
		int nivel = 9;

		List<Prueba> lp = pruebaManager.getPruebasByNivel(nivel);

		ResultActions ra =
mockMvc.perform(get("/nivel/{nivel}",nivel));
		ra.andDo(MockMvcResultHandlers.print());
		ra
		.andExpect(status().isOk())
	.andExpect(view().name("nivel"))
	.andExpect(forwardedUrl("/WEB-INF/jsp/nivel.jsp"))
	.andExpect(model().attributeExists("nivel"))
	.andExpect(model().attributeExists("pruebas"))
	.andExpect(model().attributeExists("elementos"))
	.andExpect(model().attribute("elementos", is(lp.size())))
	.andExpect(model().attribute("pruebas", hasItem(
			lp.get(0)
```

```
    }
}
```

6.2 Tests de aceptación

Para poder visualizar los tests de aceptación que fallan ha sido necesario en primera instancia configurar selenium y en segunda instancia crear un método que me permita hacer capturas de pantalla de lo que se está mostrando en selenium.

6.2.1 Configuración de selenium

Selenium es una herramienta de pruebas que me permite simular el comportamiento de una página web de modo automatizado como si fuera un humano el que realiza las acciones.

Para su correcto uso ha sido necesario usar las correspondientes librerías que han sido incluidas en el archivo pom.xml pero también ha sido necesario configurar el entorno de pruebas, para ello hay que realizar las siguientes acciones:

1. Descargar el driver de los correspondientes navegadores web (Yo he hecho las pruebas con Chrome):
    - Driver para Chrome
    - Driver para Gecko
2. Crear una variable estática a dicha ubicación:

```
public final static String CROMEPATH =
"C:/Users/ruben/Downloads/chromedriver.exe";
```

3. Configurar el driver selenium para que cargue a 600x800 píxeles usando la variable CROMEPATH:

```
//Establezco el driver
      System.setProperty("webdriver.chrome.driver",
epub.generator.stat.Constantes.CROMEPATH);

      ChromeOptions options = new ChromeOptions();
      options.addArguments("window-
size="+Constantes.KINDLEWIDTH+","+Constantes.KINDLEHEIGHT);

      DesiredCapabilities cap = DesiredCapabilities.chrome();
      cap.setCapability(ChromeOptions.CAPABILITY, options);

      WebDriver driver = new ChromeDriver(cap);
```

6.2.2 Creación de las capturas de pantalla en selenium

Crear las capturas de pantalla en selenium de los tests que fallan me va a permitir localizarlos de modo muy fácil y ver qué es lo que ocurre con dichos tests, para ello hemos implementado el siguiente método:

```
private Image getScreenshot(final WebDriver d, final WebElement e)
throws IOException {
            final BufferedImage img;
            final byte[] screengrab;

            Coordinates cor = ((Locatable) e).getCoordinates();
            cor.inViewPort();

            screengrab = ((TakesScreenshot)
d).getScreenshotAs(OutputType.BYTES);

            img = ImageIO.read(new ByteArrayInputStream(screengrab));

            return img;
      }
```

Lo importante del método es que pasa como parámetro el elemento del cual se desea hacer la captura de pantalla y va a ser importante porque mediante la función inViewPort(), me va a permitir hacer scroll en el navegador hasta dicha ubicación y por lo tanto hacer la captura del elemento que está fallando.

Lo último que nos quedará es almacenar esa captura en una ubicación correcta, y hemos decidido que esa ubicación sea dentro de la carpeta target de modo que si ejecutamos la tarea de maven clean, todas esas capturas sean borradas:

```
public final static String ACCEPTANCETESTSIMGPATH ="./target/surefire-
reports";

Image im = getScreenshot(driver, respuesta);
BufferedImage bi = (BufferedImage) im;
File f = File.createTempFile("screenshot", ".jpg",
      new File(Constantes.ACCEPTANCETESTSIMGPATH));
ImageIO.write(bi, "jpg", f);
```

6.3 Problemas en las preguntas detectados en el testing

Se han detectado varios problemas en el testing que ni siquiera era consciente de que existiesen. Estos problemas son debidos a que tal y como he descrito, la generación de pruebas es un proceso semi-automático por lo que siempre hay lugar para que aparezcan infinidad de errores. En total ha habido que modificar unas 50 preguntas (de 3800) debido a los siguientes errores:

- Se ha detectado un enlace a una página externa al hacer click en una imagen concreta de un ejercicio. El error ha sido detectado en los tests funcionales cuando hemos comprobado que todos los enlaces tengan su equivalente para volver atrás.
- Se han detectado tuplas en la base de datos que estaban mal insertadas (dos concretamente).
- Se ha detectado una imagen que es referenciada por un archivo que no pertenece a ese nivel. La pregunta 3458 hace referencia a la imagen 3480_1.png. Esto es erróneo ya que 3458 pertenece al Nivel 3 y 3480 pertenece al Nivel 4 y a pesar de que las referencias estén bien, estructuralmente no es correcto.
- Ha habido una imagen llamada flor.png que es visualizada en los documentos en un tamaño muchísimo menor que el original, por lo que fallaban los tests que me aseguran que todas las imágenes tienen por lo menos la mitad del tamaño original. Ha habido que introducir un listado de archivos ignorados para dicha validación.
- En cierto momento durante la creación del Entorno de Pruebas para la comprobación de los controladores se probó a crear como gestor de Persistencia Hibernate y se borraron las 4 tablas de las que hacemos uso en el proyecto: prueba, pregunta, pregunta_prueba y pregunta_nivel. Hubo que restaurar un backup completo en local y volver a exportar el backup únicamente de las tablas afectadas.

6.3.1 Problema de ancho excesivo de las pruebas

En estos casos se ha decidido partir la tabla de soluciones editándola a mano. Se muestra un ejemplo de una respuesta que ha habido que editar:

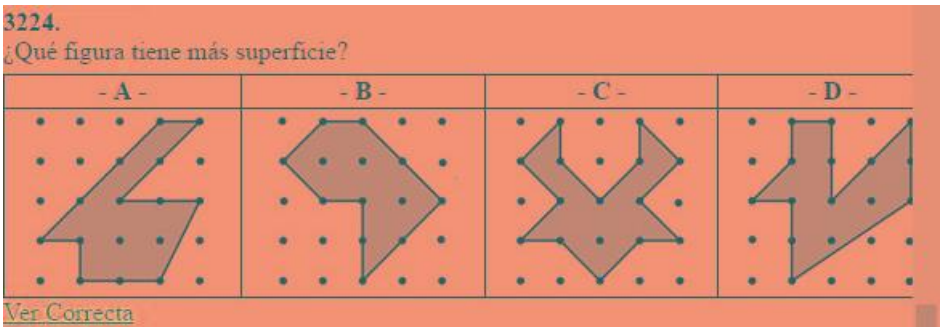
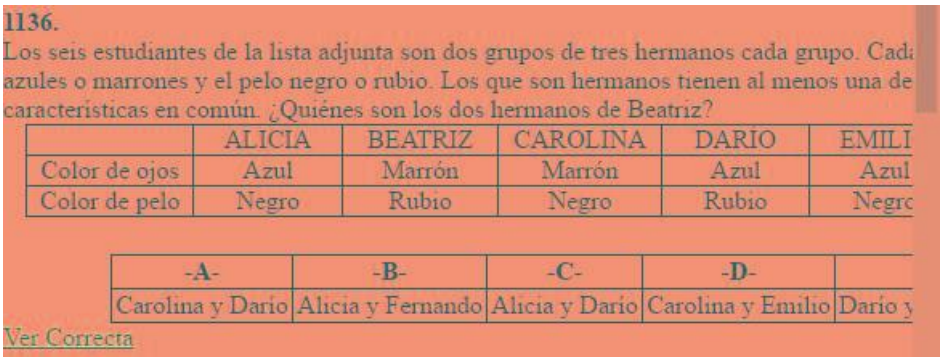


Ilustración 38 - Resultados demasiado anchos

Ha habido tablas con muchos espacios en blanco entre medio y letras distintas que ensanchan el enunciado:


1136.
Los seis estudiantes de la lista adjunta son dos grupos de tres hermanos cada grupo. Cada
azules o marrones y el pelo negro o rubio. Los que son hermanos tienen al menos una de
características en común. ¿Quiénes son los dos hermanos de Beatriz?

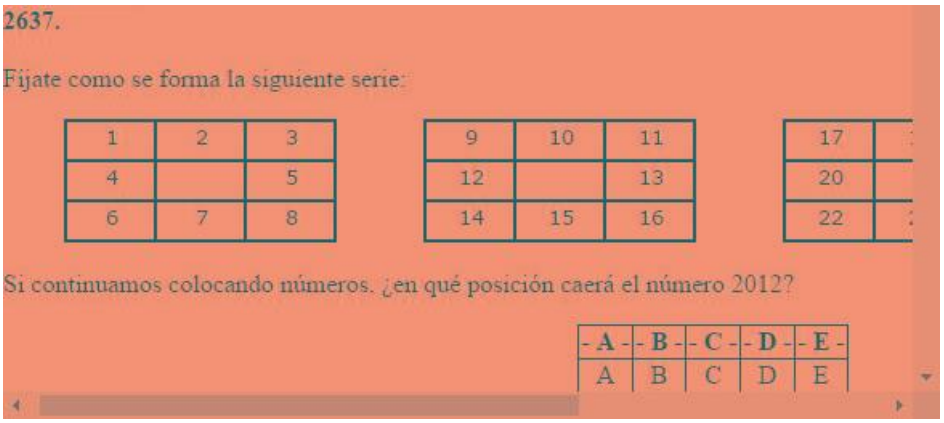
	ALICIA	BEATRIZ	CAROLINA	DARIO	EMILI
Color de ojos	Azul	Marrón	Marrón	Azul	Azul
Color de pelo	Negro	Rubio	Negro	Rubio	Negro

-A-	-B-	-C-	-D-	
Carolina y Dario	Alicia y Fernando	Alicia y Dario	Carolina y Emilio	Dario y

Ver Correcta

Ilustración 39 - Exceso de espaciado en las tablas

Enunciados demasiado anchos:


2637.

Fijate como se forma la siguiente serie:

1	2	3	9	10	11	17		
4		5	12		13	20		
6	7	8	14	15	16	22		

Si continuamos colocando números. ¿en qué posición caerá el número 2012?

- A -	- B -	- C -	- D -	- E -
A	B	C	D	E

Ilustración 40 - Enunciados demasiado anchos

6.4 Resultados de las pruebas

La aplicación ha sido probada en un alto porcentaje como muestra la imagen que muestro a continuación. Se puede observar que de las líneas de código se ha cubierto el 87,4% de las instrucciones y de las líneas de test el 89,1%. La columna de la derecha indica que hay bastantes más instrucciones de código de prueba (3350) que de código real de la aplicación (2093).

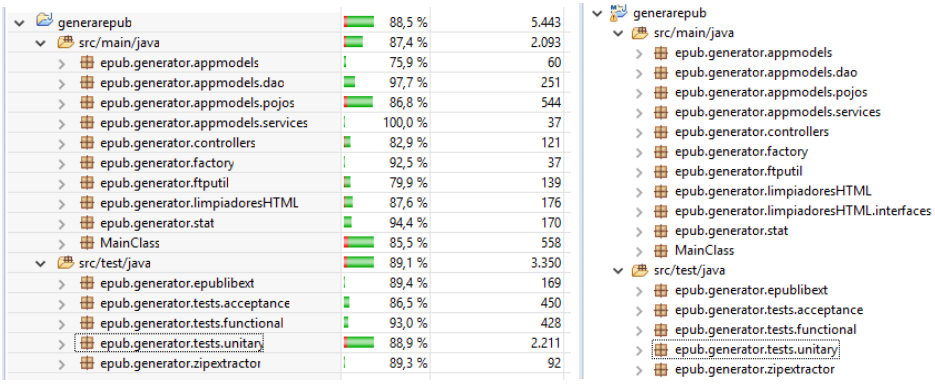


Ilustración 41 - Porcentaje de líneas de código recorrido en las pruebas

7 CONCLUSIONES

La consecución de este trabajo me ha resultado muy gratificante ya que por un lado me ha permitido aprender infinidad de nuevas tecnologías (*Ver Ilustración 35 - Gráfico de dependencias del proyecto*) trabajando sobre un proyecto al que llevo años dedicando tiempo. Además me ha permitido asegurar que el trabajo realizado hasta ahora había sido ejecutado de modo adecuado y que a pesar de que la conversión de las preguntas es un proceso semiautomático, ahora si puedo asegurar que la generación se realizaba de modo adecuado casi en su totalidad.

Adicionalmente, y en contraposición a lo que ocurre en la mayoría del mundo empresarial, este proyecto me ha permitido poder dedicarle a la creación de pruebas el esfuerzo que merece y a la edición del código también de modo que no haya ningún warning. Esto aporta una gran tranquilidad de que el software desarrollado es de calidad.

Cabría destacar también que el desacoplamiento del código fuente me ha permitido adecuarme de modo fácil a cambios que ha sido necesario introducir como se ve en el apartado *5.6*.

Me gustaría comentar también que creo que me he equivocado en la elección de entorno de Hosting la plataforma de Google Cloud ya que es una muy buena opción si planteas el proyecto desde el primer momento para ser subido a Google Cloud. Creo que si tienes ya un proyecto en formato war que es fácilmente puesto en producción en cualquier servidor apache no merece la pena complicarse y es mejor pagar un servicio de hosting especializado. Como estudiantes es un incentivo bastante grande los 300 €uros gratuitos que Google nos concede pero la realidad es que los pagas sobradamente en el tiempo que te cuesta adaptar tu aplicación para poder ser alojada allí.

Me gustaría recalcar también que el proyecto es una suma de los contenidos vistos a lo largo del curso en muchos de los distintos módulos:

- Entornos de Persistencia Especializados (impartida por Beatriz Pérez) -> La gestión de clases y entornos de persistencia
- Desarrollo de Aplicaciones para Internet (impartida por Jónathan Heras) y Entornos de Desarrollo (impartida por María Vico Pascual) ->

La inyección de dependencias y la existencia del modelo MVC con Servicios-Repositorios.

- Integración Continua (impartida por Eloy Javier Mata) la existencia de los servidores de gestión continua y múltiples plugins que me permiten llevar un seguimiento de la cantidad de código cubierta por los tests: code-coverage.

8 BIBLIOGRAFÍA

- Siriwardena, P (2014). **Mastering Apache Maven 3**. Packt Publishing. 978-1783983865
- Bharathan, R (2015). **Apache Maven CoockBook**. Packt Publishing. 978-1785286124
- Deck, P (2016). **Spring MVC: A tutorial (2ª Edición)**. Brainy Software. 978-1771970310
- Ugia Gonzalez, J y. Krishnan, S. P. T (2015). **Building Your Next Big Thing with Google Cloud Platform: A Guide for Developers and Enterprise Architects**. Apress. 978-1484210055
- Keith, M y Schincariol, M. (2013). **Pro JPA 2 (2ª Edición)**. Apress. 978-1430249269